MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2. 2016/2017

TLD7011 – LOW LEVEL DESIGN OF SOFTWARE

(All sections / Groups)

13 FEBRUARY 2017
8.00 p.m – 10.00 p.m
(2 Hours)

INSTRUCTIONS TO STUDENTS

1. This Question paper consists of 8 pages only, exclusive of the cover.

2. Attempt **ALL** out of **FOUR** questions . All questions carry equal marks and the distribution of the marks for each question is given .

3. Please print all your answers in the Answer Booklet provided.

YOU MUST ANSWER EVERY QUESTION. EACH QUESTION CARRIES THE SAME DISTRIBUTION OF MARKS.

QUESTION 1

Evaluate the task dependency diagrams shown in the figures below and answer the following questions. Assume that each node is of unit weight.

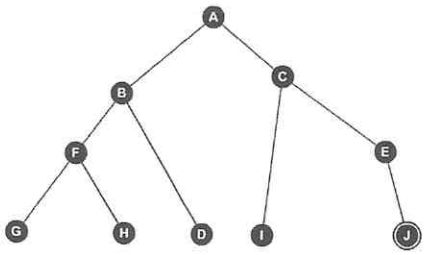


Figure A

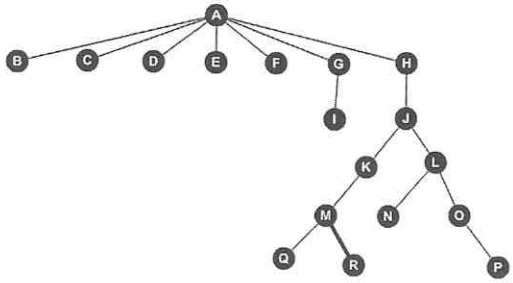
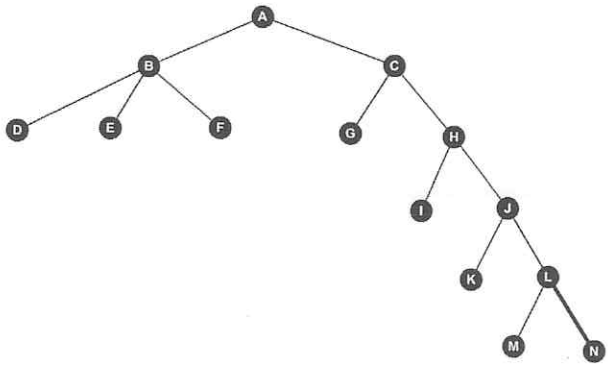


Figure B

Figure C



Figures A-C: A collection of three task dependency diagram

CONTINUED...

A. Give the average degree of concurrency for each task dependency diagram.

(6 MARKS)

B. Give the critical path length for each task dependency diagram.

(3 MARKS)

C. Name one constraint that can impact the speedup of these task dependency diagram if they were to be implemented in a parallel manner?

(1 MARK)

CONTINUED...

QUESTION 2

Dragon Inc. is one of the top toy manufacturers in China. In fact, they're a pioneer in toy manufacturing. They started production at a time when few toys were being produced commercially. Hence, they dominated the market and became the leader in the toy production industry.

Their `produceToy()` function looked like this:

```
class ToysFactory {

    public function produceToy($toyName) {

        $toy = null;

        if ('car'==$toyName) {

            $toy = new Car();

        } else if ('helicopter'==$toyName) {

            $toy = new Helicopter();

        }



        $toy->prepare();

        $toy->package();

        $toy->label();



        return $toy;

    }

}
```

CONTINUED...

Initially, they only manufactured toy `Cars` and `Helicopters`. For this simple task the function worked well and everyone was happy. But not long after that, a cool new toy, the "Jumping Frog," was introduced by the design team. It was time to change the the `produceToy()` function:

```
class ToysFactory {

    public function produceToy($toyName) {

        $toy = null;



        if ('car'==$toyName) {

            $toy = new Car();

        } else if ('helicopter'==$toyName) {

            $toy = new Helicopter();

        } else if ('jumpingFrog'==$toyName) {

            $toy = new JumpingFrog();

        }



        $toy->prepare();

        $toy->package();

        $toy->label();



        return $toy;

    }

}
```

CONTINUED...

A. What is the main problem with their approach, taking into the consideration that in the future more toys are likely to be introduced? Discuss in terms of code-reusability and exception-handling. **(3 MARKS)**

B. Refactor the produceToy() function in order to remove the constructor into its own dedicated class. **(3 MARKS)**

C. Name two design patterns that can potentially be used to replace the functionality of the ToysFactory() class, and explain their advantages in terms of code maintenance and object creation. **(4 MARKS)**

QUESTION 3

Consider the following code snippets. Answer the following questions based on these code snippets

```
public class DomainObject {

        public DomainObject (String name)      {
                _name = name;
        };

        public DomainObject () {};

        public String name ()  {
                return _name;
        };

        public String toString() {
                return _name;
        };

        protected String _name = "no name";
}
```

CONTINUED...

```
public class Movie extends DomainObject {
    public static final int  CHILDRENS = 2;
    public static final int  REGULAR = 0;
    public static final int  NEW_RELEASE = 1;


        private int _priceCode;

        public Movie(String name, int priceCode) {
                _name = name;
                _priceCode = priceCode;
        }

        public int priceCode() {
                return _priceCode;
        }

        public void persist() {
                Registrar.add ("Movies", this);
        }

        public static Movie get(String name) {
                return (Movie) Registrar.get ("Movies", name);
        }
}
```

```
class Tape extends DomainObject
    {
    public Movie movie() {
        return _movie;
    }
    public Tape(String serialNumber, Movie movie) {
        _serialNumber = serialNumber;
        _movie = movie;
    }
    private String _serialNumber;
    private Movie _movie;
    }
```

```
class Rental extends DomainObject
    {
    public int daysRented() {
        return _daysRented;
    }
    public Tape tape() {
        return _tape;
    }
    private Tape _tape;
    public Rental(Tape tape, int daysRented) {
        _tape = tape;
        _daysRented = daysRented;
    }
    private int _daysRented;
    }
```

CONTINUED...

```
class Customer extends DomainObject
    {
    public Customer(String name) {
        _name = name;
    }
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + name() + "\n";
        while (rentals.hasMoreElements()) {
            double thisAmount = 0;
            Rental each = (Rental) rentals.nextElement();

            //determine amounts for each line
            switch (each.tape().movie().priceCode()) {
                case Movie.REGULAR:
                    thisAmount += 2;
                    if (each.daysRented() > 2)
                        thisAmount += (each.daysRented() - 2) * 1.5;
                    break;
                case Movie.NEW_RELEASE:
                    thisAmount += each.daysRented() * 3;
                    break;
                case Movie.CHILDRENS:
                    thisAmount += 1.5;
                    if (each.daysRented() > 3)
                        thisAmount += (each.daysRented() - 3) * 1.5;
                    break;

            }
            totalAmount += thisAmount;

            // add frequent renter points
            frequentRenterPoints ++;
            // add bonus for a two day new release rental
            if ((each.tape().movie().priceCode() == Movie.NEW_RELEASE) &&
each.daysRented() > 1) frequentRenterPoints ++;

            //show figures for this rental
            result += "\t" + each.tape().movie().name()+ "\t" +
String.valueOf(thisAmount) + "\n";

        }
        //add footer lines
        result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
        result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent
renter points";
        return result;

    }
    public void addRental(Rental arg) {
        _rentals.addElement(arg);
    }
    public static Customer get(String name) {
        return (Customer) Registrar.get("Customers", name);
    }
    public void persist() {
        Registrar.add("Customers", this);
    }
    private Vector _rentals = new Vector();
    }
```

CONTINUED...

A. Extract the switch statements for calculating the amount of rental from the statement() method in the Customer class. What is the name of this type of refactoring? **(5 MARKS)**

B. Subsequently, move the refactored code that you have isolated in part (a) to the Rental class. Show the Rental class after you have moved the code. What must you do once you have completed the move of the code? **(5 MARKS)**

QUESTION 4

A. What are the characteristics of software projects which will benefit from using an agile approach? **(3 MARKS)**

B. Draw a diagram which shows the agile software life development cycle. Highlight the phases in which active stakeholder participation is important. **(4 MARKS)**

C. Someone who practices agile development gave the following statement.
"Code Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior. Agile teams are maintaining and extending their code a lot from iteration to iteration, and without continuous refactoring, this is hard to do." To what extent do you agree with this statement? **(3 MARKS)**

END of Paper